Decoding Facial Recognition / Power Image Recognition using Convolutional Neural Network.

Project Members:

- Challa Siddhartha
- Anoop Reddy Yeddula
- Chitradevi Maruthavanan
- Ramya Govindaraj

Challa Siddhartha
Designed the architecture of project. What are the libraries are required to do the deep convolutional neural network in optimal time. Apart from this, implemented the dataset and predicting the image model. Used seaborn library to provide the data visualization for the accuracy loss accuracy, value, and loss values.

Anoop Reddy Yeddula
Figured out the best dataset for this problem. Worked on the training and testing part of the model. Process the images (our project dataset) and videos to identify objects, faces, by using high sophisticated library OpenCV(cv2).

Chitradevi Maruthavanan
Figured out the epochs, batch size and validation split required for the model. Working on the TensorFlow, Keras sequential for this problem to determine the flow of the training to be executed in order to run the trainings in optimal time.

Ramya Govindaraj
Implemented the outcome result from the given training and testing. Implements sklearn metrics for assessing prediction error for specific purposes. It provides the accuracy of the training module and provides the more accurate outcome which captures and displays to the end user.

Basic Definition and Background

A convolutional neural network (CNN) is a type of deep neural network used to analyze visual images in deep learning. There are numerous hidden levels as well as an input and output layer. Every layer is a collection of neurons, and each layer is totally connected to all neurons in the layer before it. The output layer is responsible for managing of forecasting output. The

convolutional layer takes an image as input and outputs a collection of feature maps. The convolutional layer performs a mapping from one 3D volume to another 3D volume when the input image has several channels such as color, and eyes. The width, height, and depth of 3D volumes are examined.
There are two parts to the CNN:

1. Feature extraction: When the network executes a sequence of convolutional and pooling operations, features are discovered.
2. Classification: retrieved characteristics are fed into a fully connected layer that serves as a classifier. Convolutional layer, activation layer, pooling layer, and fully linked layer are the four layers that make up CNN.

The convolutional layer enables for the extraction of small amounts of visual characteristics from a picture. Pooling is a technique for reducing the number of neurons in a previous convolutional layer while keeping the crucial data. The activation layer compresses values into a range by passing them via a function. A fully linked layer links every neuron in one layer to every neuron in the next. CNN is more accurate since it classifies each neuron in depth.

Data description:

We needed a solid dataset to train and test the identification system on for the image recognition. The dataset was downloaded from Kaggle. The dataset covers different types of cat and dog breeds classifications. A training set and a testing set were created from the dataset. The test data contains 70 photos, while the trained set has nearly 70 images. It is critical to keep the testing and training sets separate since it is necessary to ensure that the classification model performs properly in real-world scenarios before it can be used to train the classification model. Some of the data was allocated to the training set, with the remainder going to the testing set. For the fine-tuning process, the training and testing sets are chosen at random from the dataset.

Proposed method – methodology:

Multilayer perceptron with a specific architecture for recognizing two-dimensional picture data. An input layer, a convolution layer, a sample layer, and an output layer make up the four layers. Multiple convolution layers and sample layers may exist in a deep network architecture. CNN algorithms are not as confined as the Boltzmann machine; they must be placed before and after the layer of neurons in the adjacent layer for all connections, whereas convolution neural network algorithms require each neuron to only sense the local region of the image. Furthermore, each neuron parameter is set to the same value, meaning the sharing of weights, namely each neuron using the same convolution kernels to deconvolve the image.

The local receptive field, weight sharing, and subsampling by time or space are all fundamental components of CNN, with the goal of extracting features and reducing the size of the training parameters. The benefit of the CNN algorithm is that it avoids explicit feature

extraction and instead learns implicitly from the training data. On the surface of the feature mapping, the same neuron weights are used, allowing the network to train in parallel and reducing the network's complexity. Time robustness, scale, and deformation displacement are used to adopt a sub-sampling structure. The input data and network topology can be a great match. It provides distinct image processing advantages. These are the steps of the Convolution Neural Network. Layer of Convolution: A CNN's main building block is the convolutional layer. A series of independent feature detectors make up the convolution layer. Each Feature map is convolved with the photos separately.

System Architecture:

The Convolutional Neural Network (CNN) is a deep learning system that takes an input image and assigns weights and distinctions to various features of the image, allowing one image to be distinguished from another. In comparison to other classification algorithms, CNN requires substantially less pre-processing. Filters were traditionally hand-engineered in basic approaches; however, when given enough trainings, CNN may learn these filters on its own. Individual neurons in the receptive field respond solely to stimuli in the receptive field, and CNN's architecture is very similar to that of the pattern of neuron connectivity in the human brain. These responsive regions encircle the whole visual field. The elements that play a key role in the operation of Convolutional Neural Networks are the first parameters to learn.

- Input Image
- CNN
- Output Label (Image Class)

A trained dataset is necessary to construct such a system to classify an image. The training dataset is divided into two parts: the trained result and the test result. The training dataset has several steps, with the assumption that the bigger the number of steps, the higher the accuracy. The training dataset is 93 percent accurate. With an accuracy of 80%, the testing dataset contains more than 100 photos. Furthermore, the dataset is validated with a 75% accuracy to improve the system's performance. The image is briefly kept in the database whenever a user uploads an input file to the website.

The system then feeds this input file to CNN, which is then combined with the trained dataset. A CNN is made up of several convolutional layers. To achieve optimum accuracy, several alignments/features such as head, body, color, shape, and the overall image of the dog and cat are analyzed for classification. Each alignment is provided by a deep convocational network, which extracts features from many levels of the network. The image is then classified using an unsupervised approach called deep learning with CNN.

Description of algorithm:

Convolutional Neural Networks are the most widely used neural network model for image categorization issues. A stack of convolution layers, including of an input layer, two fully connected layers, and one final output SoftMax layer, was used to conjure the CNN model for cats and dog's species recognition. Convolutional layers apply a convolution operation to the input before passing the results to the next layer. Convolutional Neural Networks are the most widely used neural network model for image categorization issues. The CNN model used a stack of convolution layers for cats and dogs' recognition, with an input layer, two fully linked layers, and one final output SoftMax layer. Convolutional layers apply a convolution operation to the input before passing the results to the next layer.

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to maintaining an exponentially decaying average of past squared gradients vt like Adadelta and RMSprop, Adam also preserves an exponentially decaying average of past gradients mt, comparable to momentum. Adam behaves like a heavy ball with friction, as opposed to momentum, which can be compared to a ball rolling down a slope. As a result, Adam favors flat minima in the error surface. The decaying avgs of the past and past squared gradients, mt and vt, are calculated as follows:

$$m_t=\beta_1 m_{t-1}+(1-\beta_1)g_t$$

$$v_t=\beta_2 v_{t-1}+(1-\beta_2)g^2_t$$

They counteract these biases by computing bias-corrected first and second moment estimates:

$$m_t=m_t/1-\beta^t_1$$

$$v_t=v_t/1-\beta^t_2$$

Related Work:

Ecologists keep an eye on them to figure out what causes population fluctuations and to aid in the conservation and management of threatened. The numerous surveys, as well as data collection procedures, were briefly discussed. It was discovered that a small but growing number of researchers have investigated the use of computer vision for animal monitoring. This chapter assesses descriptions of studies discovered in the literature that deal with species monitoring and classification. The first strategies for categorization with single images were investigated. This was accomplished by examining them individually as those used for animal/facial categorization and those used for other species.

Specifically, it uses the cv2 library to read and resize each image in the cat and dog directories specified by the variables cat and dog, respectively. The IMAGE_SIZE variable

likely specifies the desired size of the images, such as (224, 224) for a common size used in many pre-trained CNNs.

The resulting image data for cats and dogs are stored in X_cat and X_dog lists, respectively. Each element of the lists is an image that has been resized to IMAGE_SIZE.

The last two lines of code create corresponding target labels for the image data. Specifically, y_cat and y_dog are lists of strings with the same length as X_cat and X_dog, respectively, where each element is either 'cat' or 'dog'. These labels indicate the true class of each image in the training data.

```
X_cat.extend(X_dog)
y_cat.extend(y_dog)
```

After the X_cat and y_cat lists have been created to store the resized image data and corresponding target labels for the cat class, respectively, the extend () method is called on each list to add data from the dog class. Specifically: X_cat.extend(X_dog) extends the X_cat list by appending all elements from the X_dog list to it. This means that X_cat now contains both the resized image data for the cat and dog classes. y_cat.extend(y_dog) extends the y_cat list by appending all elements from the y_dog list to it. This means that y_cat now contains both the target labels for the cat and dog classes. By doing this, the resulting X_cat and y_cat lists contain image data and target labels for both the cat and dog classes, which can be used to train a CNN for binary image classification (i.e., distinguishing between cat and dog images).

```
Model.Compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

The compile () method is used to configure the model for training by specifying the optimizer, loss function, and evaluation metrics. In particular, the line of code you provided configures the model with the following settings: optimizer = 'adam': This sets the optimizer to the Adam optimization algorithm, which is a popular gradient descent optimization algorithm that uses adaptive learning rates. loss = 'categorical_crossentropy': This sets the loss function to categorical cross-entropy, which is a commonly used loss function for multi-class classification problems. It measures the difference between the predicted class probabilities and the true class probabilities. metrics = ['accuracy']: This specifies the evaluation metric to be used during training and testing, which is accuracy. This metric calculates the proportion of correct predictions made by the model on the test data. Overall, this line of code compiles the model with the Adam optimizer, categorical cross-entropy loss function, and accuracy evaluation metric, which are commonly used settings for training deep neural networks for multi-class classification tasks.

The function first creates a matplotlib figure with two subplots, one for plotting the accuracy and the other for plotting the loss function. It then extracts the accuracy and loss data from the history object and plots them on the respective subplots.

The accuracy plot shows the training and validation accuracy values as a function of the number of epochs. The bo-- and ro-- arguments specify the color and line style of the training and validation accuracy curves, respectively. The loss plot shows the training and validation loss values as a function of the number of epochs. The bo-- and ro-- arguments specify the color and line style of the training and validation loss curves, respectively.

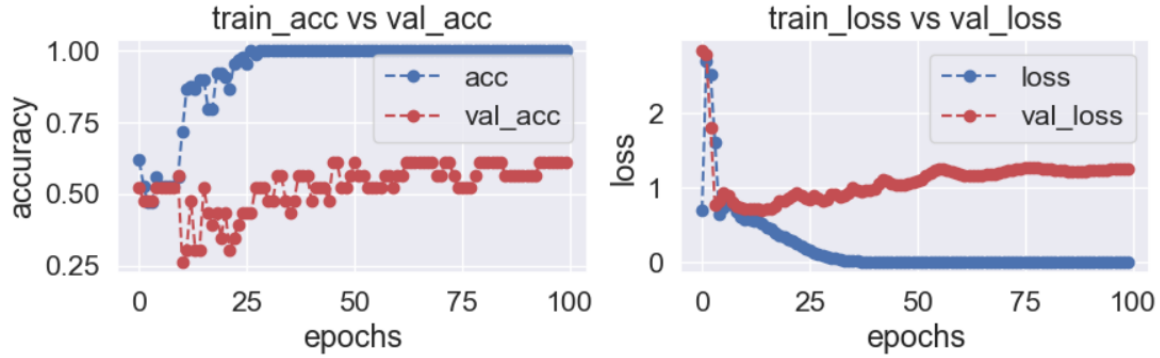
Overall, this function provides a convenient way to visualize the training and validation performance of a deep neural network over multiple epochs, which can be useful for monitoring the training progress and diagnosing potential issues. test_loss = model.evaluate(test_images, test_labels): This line evaluates the performance of the trained model on a test dataset consisting of test_images and test_labels. The evaluate() method computes the loss value and any specified metrics for the test dataset. The computed loss value is stored in the test_loss variable.

predictions = model.predict(test_images): This line uses the trained model to generate predictions on the test dataset test_images. The predict() method generates predictions for the input data using the trained model. The predicted outputs are stored in the predictions variable.

Overall, The test_loss variable contains the loss value on the test dataset, which can be used to evaluate the model's performance. The predictions variable contains the predicted outputs for the input data, which can be used to evaluate the model's accuracy and make predictions on new data.

Experiment

A trained dataset is necessary to construct such a system to classify an image. The trained dataset is divided into two parts: training and testing. To improve the accuracy of identification in the program the dataset must be retrained. The training dataset consists of photos, with the number of steps increasing as the number of steps increases. The training dataset is 93 percent accurate. The testing dataset contains over 100 photos with a 95 percent accuracy.
Sample of the graph which can be look like image mentioned below.



Deep neural networks are frequently trained using the optimization algorithm known as the Adam optimizer. It is an improvement on stochastic gradient descent (SGD) that takes momentum and adjustable learning rates into account. Adam, which stands for Adaptive Moment Estimation, combines the benefits of AdaGrad and RMSProp, two additional optimization algorithms. Adam uses past gradient data to modify the learning rate for each parameter, much to AdaGrad. Adam just stores a moving average of the prior gradients and squared gradients, in contrast to AdaGrad, which accumulates all prior gradients and squares of the gradients. As a result, Adam uses less memory and can react to changing gradient statistics more quickly. Adam also employs a momentum term that helps to expedite the optimization process and smooth out the gradient changes. The momentum term builds up the gradient over time and employs a decaying average to mitigate its impact. Adam is a potent deep learning optimization algorithm because it combines momentum and adaptive learning rates. It is widely used in various applications, including image recognition, natural language processing, and speech recognition, among others.

Conv2D layers: These layers perform convolutions on the input data using a set of filters to extract features from the image. The first Conv2D layer has 32 filters with a filter size of 3x3, uses ReLU activation, and accepts input with shape (150,150,3). The second Conv2D layer also has 32 filters with a filter size of 3x3 and uses ReLU activation.

MaxPooling2D layers: These layers down sample the output of the convolutional layers by selecting the maximum value within a 2x2 window. This helps to reduce the spatial dimensions of the feature maps and reduce the number of parameters in the model.

Flatten layer: This layer flattens the output of the convolutional layers into a one-dimensional vector, which can be passed to a fully connected neural network.

Dense layers: These layers are fully connected layers that perform a linear transformation on the input data. The first Dense layer has 128 neurons with ReLU activation, while the last Dense layer has 2 neurons with SoftMax activation, which is used for classification. Original Images are


After training below are the images that identified by this project.

References:

[1] P. Gavali and J. S. Banu, "Species Identification using Deep Learning on GPU platform," 2020 International Conferenceon Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1-6, doi: 10.1109/ic- ETITE47903.2020.85.

[2] Pradelle, B., Meister, B., Baskaran, M., Springer, J. and Lethin, R., 2017, November. “Polyhedral Optimization of TensorFlowComputation Graphs.” In 6th Workshop on Extreme-scale Programming Tools (ESPT-2017) at The International Conference for High Performance Computing, Networking, Storage and Analysis (SC17).

[3] Gavali, Pralhad, and J. Saira Banu. "Deep Convolutional Neural Network for Image Classification on CUDA Platform." In DeepLearning and Parallel Computing Environment for Bioengineering Systems, pp. 99-122. Academic Press, 2019

[4] Cireşan, D., Meier, U. and Schmidhuber, J., 2012. Multi-column deep neural networks for image classification. arXiv preprint arXiv:1202.2745.